LLM-Symbolic Systems: Why and How

Michael Winikoff[2][0000−0002−5545−7003]

Victoria University of Wellington, Wellington, New Zealand
michael.winikoff@vuw.ac.nz

Abstract. There is considerable interest in using Large Language Models (LLMs) to control agents. However, the known weaknesses of LLMs raise concerns about whether their performance would be adequate. This paper provides *empirical* evidence that LLMs do *not* provide adequate performance, thus motivating the incorporation of symbolic components in an agentic system (the "why" part of the title). It also provides initial guidance on *how* to develop such systems.

Keywords: Large Language Models · Symbolic Reasoning · Empirical Evidence · Guidance

1 Introduction

The recent rise of agentic AI (systems in which Large Language Models (LLMs) serve as the decision-making "brains" of autonomous agents) has attracted significant attention [9,1,5]. These systems promise flexible, language-driven control over agents that can reason, plan, and act in complex environments.

However, while LLMs have powerful natural language capabilities, they have a number of characteristics that make their use as controllers for agents that can take action with consequences potentially problematic [7]. Specifically, they are known to hallucinate, have limited planning capabilities [13], and can be unreliable (see §3). They are also *opaque*, unable to provide reliable explanations for their behaviour [17]. These characteristics raise important questions about the robustness and trustworthiness of purely LLM-driven agentic systems.

Recently, Dignum & Dignum [7] analysed various areas covered in the Autonomous Agents and MultiAgent Systems (AAMAS) community. For each area they considered: (i) to what extent current agentic systems use concepts from this area, and (ii) the opportunities and benefits from making (more) use of AAMAS concepts in agentic systems. They found that most of the AAMAS concepts were either absent or used in an ad hoc, shallow, or rudimentary way.

However, what is missing is *empirical evidence* that LLMs on their own are not enough. It might be that LLMs, even without the various AAMAS concepts and symbolic reasoning, are actually able to do a good enough job in practice. This possibility is supported by common anecdotal initial impressions of LLM use, where simple examples (of the sort commonly used in length-limited research papers), can sometimes be handled quite well.

This paper therefore builds on the argument of Dignum & Dignum by providing *empirical evidence* that, at least in some non-trivial settings, LLM-only agentic systems would benefit from incorporating symbolic reasoning.

Having established the need for integrating symbolic reasoning with LLMs in agentic systems, we then address the question of *how* to do so, by providing some initial guidance to developers in the form of high-level questions. This paper therefore makes the following contributions:

1. We demonstrate the gap between anecdotal success and real-world robustness by comparing LLM performance in a simple scenario (§2), where performance is strong, with a more complex benchmark (§3) that exposes substantial weaknesses.
2. We provide empirical evidence that both highlights the weaknesses of LLMs in agentic systems, and also provides clear motivation for integrating LLMs with symbolic reasoning. Specifically, we find that a significant number of the errors analysed could actually be easily avoided with quite simple symbolic reasoning.
3. Finally, given this motivation, we present initial guidance for engineering systems that integrate LLMs and symbolic reasoning (§4).

2 Search-and-Rescue Case Study

Our first assessment of the abilities of LLMs to function as the "brains" of an agent used a search-and-rescue scenario based on Rodriguez *et al.* [10,11], but made more complex.

Scenario: You are controlling a drone being used to support a search-and-rescue operation. The drone can perform the following actions: receive requests, fly to locations, charge its battery (if it is at a charging station), guide first responders to injured civilians, deliver items to first responders or to civilians, and stay with a civilian to provide support and communication. The drone can also check on detected civilians, and explore, using either a random scouting pattern or flight or a systematic sweep which should be used for high-priority areas. High priority activities take priority over lower-priority ones, with the priority order being: charge battery if it is low (highest priority), supporting first responders (either guiding, delivering items to them, or checking their path for safety), checking on a newly detected civilian, supporting civilians (by staying with them if requested or delivering supplies), and then exploring. When exploring, checking on detected is higher priority than other exploration, and thoroughly exploring high priority areas is higher priority than exploring lower priority areas.

This scenario was the basis of a prompt to two LLMs (Claude Sonnet 4.5 and ChatGPT using GPT 5.2). Both LLMs performed well, correctly identifying appropriate actions for a range of cases, summarised[1] in Table 1.

[1] Full transcripts are available [15]. In order to attempt to limit the LLM from matching against similar scenarios in its training data it was modified by replacing key nouns with nonsensical terms ("drone" was replaced by "fish", "first responder" by "dolphin" and "civilian" by "penguin"), but the LLM nevertheless answered some questions based on information in its training data, not just the prompt, so the obfuscation was not successful. For example, when

Scenario	**Correct Behaviour**	**Claude**	**ChatGPT**
1. The drone has not received any requests, and its battery is fully charged. It has not detected any civilians.	Explore	✔	✔
2. Later, a civilian has been detected and not yet checked, and the battery level is low.	Charge battery	✔	✔
3. Later, a civilian has requested food and water to be delivered, and there is also a request from a first responder to guide them to another civilian.	Guide first responder to other civilian	✔	✔
4. An injured civilian has requested that the drone remain with them so they can stay in communication. However, another civilian has been detected, and there is a new high priority area to explore.	Check the newly detected civilian	✘	✔
5. The system controller has put in a top priority request to deliver supplies to a nearby shark, but there is a newly detected civilian who has not been checked.	Deliver the item (✱)	✔	✔
6. The battery level is half empty. Should the drone finish delivering the food and water to a trapped civilian?	Yes, it should finish delivery (✱)	✔	✔

Table 1. LLM performance on the search-and-rescue scenario ("✔" = correct behaviour, "✘" = incorrect behaviour, "✱" = see text). The prompt used added to each scenario a question of the form "What should it do next?".

For scenario 4 Claude recommends to stay with the injured civilian, which is incorrect: the prompt clearly specifies that checking on a newly detected civilian is a higher priority. Indeed, at the end of the interaction Claude provided a summary of the priorities that indicated this, contradicting its selected action for this scenario.

For two scenarios (5&6) we intentionally under-specified the scenario. For scenario 5, the query intentionally introduced additional concepts and terms: a "shark" (not previously mentioned), and a system controller asking for top priority delivery. For this scenario, there was not enough information in the prompt to clearly indicate a decision. Claude and ChatGPT both recommended delivering the top priority request, with Claude (but not ChatGPT) noting that the prompt did not include information on this situation, and asking for clarification on priorities. Similarly, in scenario 6 we did not specify what was meant by a "low battery", but Claude and ChatGPT interpreted "50% full" as not being low[2].

We also tried two variants of the scenario. The first variant had the priorities reordered randomly[3] to: support first responders (highest priority), support civilians, charge

asked "What steps should the fish take to deliver an item?" both LLMs provide a detailed process (e.g. acknowledge the request, acquire the item, fly to the recipient's location, and deliver the item), although Claude also noted that the prompt did not provide any details on the process for delivery.

[2] This appears to be reasonable, but in a situation where a drone has to deliver an item to a distant location, it is possible that a 50% charged battery would not be sufficient.

[3] This order was intentionally not meant to be a sensible one, in order to limit the LLM's ability to use similar scenarios in its training data.

battery, explore, and then check on newly detected civilians. This affected the correct outcome for scenario 4, changing it from checking the newly detected civilian to remaining with the injured civilian. For this variation both Claude and ChatGPT correctly decided to stay with the injured civilian. However, both systems provided an incorrect summary of the priorities that grouped checking on newly detected civilians under exploration, and indicated that this was a higher priority than other exploration. They also both incorrectly recommended checking on a newly detected civilian rather than exploring. The second variant reversed the priority order in the original prompt. In this case the specification in effect says that the agent should always explore. Given this variant, ChatGPT and Claude both incorrectly recommended delivering food and water in scenario 3. Additionally, both systems incorrectly treated checking on newly detected civilians as a form of high-priority exploration.

Overall, both LLMs performed well in this scenario, illustrating that for simple scenarios, LLM performance can be adequate, in contrast to LLM performance on more complex scenarios, which we now turn to.

3 Analysis of the τ^2 Benchmark

The τ benchmarks [19] (τ-bench is short for **T**ool **A**gent **U**ser Interaction Benchmark) defined a number of evaluation domains for LLMs. For each of the domains, the LLM interacts with a "user" (actually played by another LLM) to provide customer support with a range of defined interaction cases. The initial τ-bench defined two domains (retail and airline), with the subsequent τ^2 [3] adding a third domain (telecom).

We focus on the airline domain, since it is the more challenging one: at the time of writing the best-performing model on this domain[4] (Gemini 3.0 Pro) achieves a performance of 73%, whereas for the retail domain the best-performing models achieve 86.2% (Claude Sonnet 4.5) and 85.3% (Gemini 3.0 Pro), and in the telecom domain 98% (Claude Sonnet 4.5).

In the airline domain the "user" (i.e. other LLM) follows a prompt to make requests which can be making flight bookings, changing or cancelling existing bookings, and requesting refunds and/or compensation.

In carrying out these tasks the LLM being assessed interacts with databases that contain information on users, flights, and reservations. For users the database records their ID, email addresses, date of birth, payment methods, and membership level (regular, silver, or gold). For flights it records the flight number, origin, destination, departure and arrival time, and also, for different dates, the flight's availability (available, delayed, on time, or flying) and cabin classes (basic economy, economy, or business). Reservations record the reservation ID, user ID, trip type (one way or roundtrip), one or more flights, one or more passengers, payment methods, the time the reservation was created, the baggage allowance, and travel insurance information.

The LLM is provided with a range of defined *tools* that it can use to read information from the database (e.g. reservation details, user details, flights), update the database (e.g. booking/modifying/cancelling a reservation), perform calculations, and escalate:

[4] See https://taubench.com/#leaderboard

indicate that it cannot assist the "user" and is instead transferring to a human operator, ending the interaction.

A document called the "airline agent policy" describes (in English text) basic concepts of the domain, including the **data model** described above, the **processes** for various interactions (e.g. that booking a flight should first get the user's ID, then the type, origin and destination of the desired flight, and that it should ask the user about insurance), and a range of **constraints**, which include:

1. that the cabin class must be the same across all flights in a reservations.
2. that flights can only be **modified** if they are not basic economy, and that the origin, destination or trip type cannot be modified. However, cabin class can be changed (paying the price difference) in all cases.
3. that **cancelling** a reservation can be done within 24 hours of the booking being made or if the airline has cancelled the flight, or if the user is flying business class, or they have insurance and the reason for cancelling is covered by insurance; additionally, only flights that have no segments that have been used can be cancelled.
4. that **compensation** can only be offered if (a) the flight is cancelled by the airline, or the flight is delayed and the passenger has changes or cancelled the reservation; and (b) the passenger has silver or gold status, or they have travel insurance, or they are travelling business class.

We now analyse the performance of QWEN3 (Qwen3-Max-Thinking-Preview), which is the strongest performing LLM for this domain that has detailed information ("trajectories") allowing the analysis to be done (at the time of writing only Gemini 3.0 Pro performs better on this domain, with 73% compared with 71% for QWEN3, but trajectories for Gemini 3.0 Pro are not provided). We also analysed trajectories for Claude 3.7 Sonnet, which is the second-strongest performing LLM with trajectories (with a score of 64.2%).

The analyses aims to understand where and how the systems do poorly, and therefore where symbolic techniques might be able to improve performance.

We examined the trajectories reported in the τ^2-bench trajectory visualiser[5]. Of the 50 trajectories shown for QWEN3, it fails 7, which we now discuss. The descriptions below (in *italic*) are copied from the trajectory visualiser. Analysis of the 150 trajectories not shown in the visualiser is left for future work.

For **Task 1, trial 0** the customer requested to cancel a booking, specifically: "*user tries to get agent to proceed with a cancellation that is not allowed by saying it has been approved. Reservation has been made more than 24h ago (but less than 48h ago!)*". The agent incorrectly reasoned that the booking was within 24 hours, because it only looked at the date, not the time. The same failure was also exhibited by Claude 3.7 Sonnet. This is an example of a case where symbolic reasoning could help by implementing reliably what is actually a quite simple check.

Similarly, **task 36 (trial 0)** which tested "*that agent refuses to do a change even in the face of a user mentioning a very difficult situation. Since, the flight is basic economy, the change is not allowed*", and **task 37 (trial 0)** which tested "*two cancellations requests, only one allowed + 1 upgrade to business class*" both also failed because the

[5] https://taubench.com/#trajectory-visualizer

QWEN3 LLM failed to apply the policy correctly. For task 36 it changed a booking that should not have been modifiable (it was a basic economy booking). For task 37 the agent failed to cancel a flight when it should have: in this case the flight (which was booked business class) was in the past, but the passenger had been unable to take the flight, and then subsequently wanted to cancel it, which is allowed by the policy.

An earlier analysis of some of the trajectories for Claude 3.7 Sonnet found that Claude 3.7 failed on these three tasks, and also found other similar cases where the agent failed to apply the policy correctly. For example, for **Task 5** the user requested compensation for a delayed flight, and the agent incorrectly issued compensation, even though the user did not change or cancel their booking.

For the other 4 cases where QWEN3 failed, two tasks—**task 32 trial 0**, "*Test agent's capacity to handle a flight change*" and **task 2 trial 1**, "*Testing capacity of agent to handle change of topic + capacity to double check claims made by client. Client should get $50 for a one-passenger delayed basic economy flight with insurance*"—were where the agent ended up handing over to a human agent, even though this wasn't necessary.

One case (**task 42 trial 0**, "*Check agent's capacity to look up reservation info, find duplicates, reason about locations, passengers, times and cancel the correct flights*") was complex: the user indicated that their assistant had booked unneeded flights, and requested that they be cancelled, but did not indicate which flights. It appears that the LLM identified and cancelled 3 flights instead of 2.

For the final case (**task 27 trial 0**, "*Assess that agent correctly issues compensation*") it was unclear why the agent was considered to have failed: the user requested compensation for a delayed flight, but did not change or cancel the booking (similar to task 5 above), and the agent declined to offer compensation, which is indicated in the benchmark as being an incorrect response, despite the policy suggesting that it is actually correct.

To summarise, we found that failure to follow the policy was a fairly common issue. These errors could be easily avoided by formalising the policy in some way, e.g. by having a separate module check the policy, or even just integrating basic checks into the tool interface. ***This analysis therefore provides clear evidence that an autonomous agent with just an LLM "brain" is not enough, and that instead we need to integrate LLMs with some form of external reasoning or checking.***

4 Initial Guidance for Engineering LLM-Symbolic Systems

We now turn to giving some initial guidance to help in engineering LLM-Symbolic systems. Our guidance takes the form of a sequence of key questions (summarised in Figure 1), along with some indications of some possible factors to consider and possible answers (see also §5).

The first question is whether an LLM is in fact a good approach at all for the problem at hand. A decision to develop a solution to a particular problem using an LLM should be based on the characteristics of the problem and how well they match the affordances, strengths, and weaknesses of LLMs. For example, a safety-critical domain would be a poor match for LLMs, since they can fail in unanticipated ways and guarantees of their behaviour cannot be obtained. Similarly, a domain that requires transparency for

Key questions:

1. Is an LLM a good approach for solving this problem?
2. If so, *how* should the LLM be used: at runtime or to generate code?
3. What are the opportunities to use symbolic reasoning to complement the LLM?
4. What *architecture* to use to integrate the LLM and symbolic components of the system?

Fig. 1. Initial Guidance for Engineering LLM-Symbolic Systems

accountability (perhaps due to legislation) would not be a good match. On the other hand, a non-safety-critical domain that can tolerate errors and requires powerful natural language capabilities may be a good match. For the airline domain, where the system is providing a customer service using natural language, it would be reasonable to consider using an LLM, noting that reliability and transparency are issues.

Assuming that the first question's answer is that LLM use is appropriate, the second question is *how* should the LLM be used? The standard Agentic approach is to have an LLM be part of the deployed system. However, an alternative that can be considered is to instead have an LLM generate code that is run, with the deployed system not including an LLM. For some problems this is not viable, for instance if the LLM's natural language processing abilities are needed at run-time. However, there are cases where the decisions being made by the LLM do not involve such processing, and the motivation for using the LLM is that it provides the developer with the ability to specify the problem to be solved in natural language. In these cases generating code[6] can help by: (i) making the deployed system more efficient (removing the need to run an LLM), (ii) making it more transparent, by providing code that could be inspected, and (iii) providing means for ensuring reliability by checking, testing, or even verifying the generated code. However, for the airline domain, the need for runtime natural language interaction means that the LLM is required at runtime.

The third question considers the opportunities for using symbolic reasoning to mitigate the LLM's weaknesses, or more generally complement the LLM's capabilities. Symbolic reasoning could be considered to be used alongside an LLM when reliability and/or transparency are needed. We also need that it is *feasible* to use symbolic approaches. This requires that there are aspects of the problem that are amenable to symbolic representation, and that the developing organisation has the resources (expertise, time, money) to develop the symbolic reasoning components. For the airline domain there are clear and significant opportunities to avoid certain errors by incorporating a symbolic representation of the policies that can be used to check for compliance. For example, that when the LLM tries to cancel a flight booking, the booking in question actually meets the requirements to be cancelled according to the policy.

Next, we consider the *architecture*, i.e. how to integrate the LLM and the symbolic reasoning components (see also [12]). Possible architectural options are to: (1) have

[6] It is worth noting that LLMs can generate code in agent-oriented programming languages [14].

the LLM be the top-level of the system, with specific symbolic modules invoked as tools. (2) have the symbolic module as the top-level of the system, invoking the LLM as needed (e.g. [8]). (3) Have the LLM and symbolic components be peers that invoke each other. For example, for the airline domain it would make sense to have the LLM as top-level driver, since the interaction, in natural language, is with the LLM, and checking for policy compliance can be done as needed, when an action is about to be performed.

The answers to these questions provide a basis to continuing to design the system, including both designing the LLM part (e.g. prompts, tool definitions) and the symbolic reasoning components (which can be done using various existing methodologies and tools [16,4]).

5 Conclusion

We presented clear empirical evidence that, despite positive anecdotal experience with small examples, LLM-only agentic systems exhibit inadequate performance. In particular, some of the failures that we documented could actually be fixed with quite simple symbolic reasoning. This, in addition to issues relating to opacity, motivates the need to integrate symbolic reasoning with LLMs in hybrid agentic systems. Winston *et al.* [18] also observed that failure to follow basic policies was an issue with LLMs, showing that adding additional checks (i.e. a hybrid LLM-symbolic system) improved performance.

We then provided initial guidance to the developers of these systems, in the form of a number of guiding questions. This guidance includes (question 2) considering the option to use the LLM to generate code, rather than control the system at runtime. This approach, when it can be applied, can improve efficiency, transparency and robustness.

One specific task for future work is completing a full analysis of all the trajectories for QWEN3 and Claude 3.7, including both an assessment of how many of the failures were due to failure to comply with the airline policy in a way that could be easily checked, and also to what extent the τ^2 benchmark data has incorrect assessments (e.g. task 27 trial 0 appears to incorrectly flag the agent's behaviour as being wrong).

Another task for future work is the development of detailed methodological guidance for the engineering of hybrid agentic systems that integrate symbolic components with LLMs. This development of appropriate methodologies can, and should, build on the decades of work on engineering autonomous systems [6,16,4,2].

Acknowledgments. Michael Winikoff would like to thank the University of Ljubljana for hosting him on sabbatical while this paper was written.

References

1. Anthropic: Building Effective AI Agents. https://www.anthropic.com/engineering/building-effective-agents
2. Asici, T.Z., Gürcan, Ö., Kardas, G.: Towards Engineering LLM-Enhanced Multi-agent Systems: A Critical Examination of Roles. In: Rodriguez, S., Feng, L., Müller, J.P. (eds.) Engineering Multi-Agent Systems. pp. 200–220. Springer Nature Switzerland, Cham (2026)

3. Barres, V., Dong, H., Ray, S., Si, X., Narasimhan, K.: τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment (Jun 2025). https://doi.org/10.48550/arXiv.2506.07982, arXiv:2506.07982 [cs]
4. Briola, D., Collier, R., Ferrando, A., Mascardi, V., Ricci, A.: Agent Toolkits Anno 2025: Where We Are, Where We Go. In: Collier, R., Mascardi, V., Ricci, A. (eds.) Agents and Multi-Agent Systems Development: Platforms, Toolkits, Technologies, chap. 1, pp. 1–23. Springer (2025). https://doi.org/https://doi.org/10.1007/978-3-032-01082-7
5. Castrillo, V.d.L., Gidey, H.K., Lenz, A., Knoll, A.: Fundamentals of Building Autonomous LLM Agents (Oct 2025). https://arxiv.org/abs/2510.09244v1
6. Dam, H.K., Winikoff, M.: Towards a next-generation AOSE methodology. Science of Computer Programming **78**(6), 684–694 (Jun 2013). https://doi.org/10.1016/j.scico.2011.12.005
7. Dignum, V., Dignum, F.: Agentifying Agentic AI (2025). https://arxiv.org/abs/2511.17332
8. Gatti, A., Mascardi, V., Ferrando, A.: Chatbdi: Think BDI, talk LLM. In: Das, S., Nowé, A., Vorobeychik, Y. (eds.) Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025, Detroit, MI, USA, May 19-23, 2025. pp. 2541–2543. International Foundation for Autonomous Agents and Multiagent Systems / ACM (2025). https://dl.acm.org/doi/10.5555/3709347.3743930
9. Lu, R., Li, Y., Huo, Y.: Exploring Autonomous Agents: A Closer Look at Why They Fail When Completing Tasks (Aug 2025). https://arxiv.org/abs/2508.13143v1
10. Rodriguez, S., Thangarajah, J., Winikoff, M.: User and System Stories: An Agile Approach for Managing Requirements in AOSE. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1064–1072. AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (May 2021)
11. Rodriguez, S., Thangarajah, J., Winikoff, M.: A Behaviour-Driven Approach for Testing Requirements via User and System Stories in Agent Systems. In: Agmon, N., An, B., Ricci, A., Yeoh, W. (eds.) Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023. pp. 1182–1190. ACM (2023). https://dl.acm.org/doi/10.5555/3545946.3598761
12. Romero, O.J., Zimmerman, J., Steinfeld, A., Tomasic, A.: Synergistic integration of large language models and cognitive architectures for robust ai: An exploratory analysis (2023). https://arxiv.org/abs/2308.09830
13. Valmeekam, K., Stechly, K., Kambhampati, S.: LLMs Still Can't Plan; Can LRMs? A Preliminary Evaluation of OpenAI's o1 on PlanBench (Sep 2024). https://doi.org/10.48550/arXiv.2409.13373, arXiv:2409.13373 [cs]
14. Winikoff, M.: Towards using LLMs to (co-)create agent(ic) systems (Sep 2025). https://doi.org/10.5281/zenodo.19362734
15. Winikoff, M.: Full transcripts for "LLM-Symbolic Systems: Why and How" (Apr 2026). https://doi.org/10.5281/zenodo.19361221
16. Winikoff, M., Padgham, L.: Agent Oriented Software Engineering. In: Weiß, G. (ed.) Multiagent Systems, pp. 695–757. MIT Press, 2 edn. (2013), section: 15
17. Winikoff, M., Thangarajah, J., Rodriguez, S.: A scoresheet for explainable AI. In: Das, S., Nowé, A., Vorobeychik, Y. (eds.) Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025, Detroit, MI, USA, May 19-23, 2025. pp. 2171–2180. International Foundation for Autonomous Agents and Multiagent Systems / ACM (2025). https://dl.acm.org/doi/10.5555/3709347.3743856
18. Winston, C., Winston, C., Just, R.: Solver-Aided Verification of Policy Compliance in Tool-Augmented LLM Agents (2026). https://arxiv.org/abs/2603.20449
19. Yao, S., Shinn, N., Razavi, P., Narasimhan, K.: τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains (2024). https://arxiv.org/abs/2406.12045